CONTENTS INCLUDE:

Spring Configuration

By Craig Walls

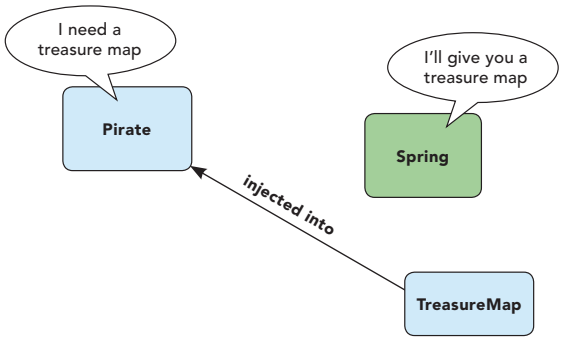
ABOUT SPRING CONFIGURATION

The Spring Framework has forever changed the face of enterprise Java development, making it easier than ever to configure and assemble application objects and services in a loosely-coupled manner. As you develop your Spring-enabled applications, you'll find this reference card to be a handy resource for Spring context configuration. It catalogs the XML elements available as of Spring 2.5, highlighting the most commonly used elements. In addition to Spring XML configuration, there'll also be a guide to Spring's rich set of annotations, which are useful for minimizing the amount of XML needed to configure Spring.

DEPENDENCY INJECTION IN A NUTSHELL

Although the Spring Framework does many things, dependency injection is the foundational functionality provided by the Spring container.

Any non-trivial application is composed of two or more objects that collaborate to perform some business logic. Traditionally, each of those objects is responsible for obtaining references to those objects that it collaborates with (its dependencies). This leads to tightly-coupled and hard-to-test code.



With dependency injection, however, objects are given their dependencies by some external entity. In other words, dependencies are injected into the objects that need them. In the case of a Spring-enabled application, it is the Spring container that injects objects into the objects that depend on them.

CONFIGURING SPRING WITH XML

As of Spring 2.0, you are encouraged to use Spring's XML Schema-based configuration, which is more flexible than the legacy DTD-based XML. A typical Spring 2.5 configuration will have, at minimum, the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/
schema/beans
       http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd">

  <!-- place configuration details here -->

</beans>
```

Within the <beans> element, you'll place bean declarations and other elements that configure your application's context. The "beans" namespace was the first and is still the primary namespace in Spring's XML configuration—but it isn't alone. Spring also comes with seven more namespaces that will be described in this reference card. If you wish to use one of the other namespaces, you'll need to be sure to declare them. For example, if you want to use the "context" namespace, you should declare it in XML as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/
context"
   xsi:schemaLocation="http://www.springframework.org/
schema/beans
       http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/
spring-context-2.5.xsd">

  <!-- place configuration details here -->

</beans>
```


Spring Configuration

www.dzone.com

THE BEANS NAMESPACE

Schema URI
www.springframework.org/schema/beans

Schema XSD
www.springframework.org/schema/beans/spring-beans-2.5.xsd

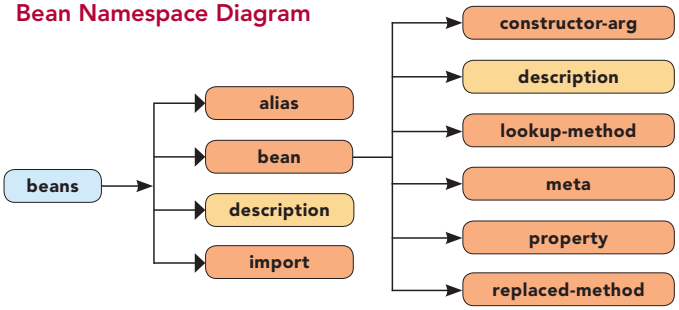
The beans namespace is the core Spring namespace and the one you'll use most when configuring Spring. The root element is the <beans> element. It typically contains one or more <bean> elements, but it may include elements from other namespaces and may not even include a <bean> element at all.

Spring XML Diagram Key

The Spring XML diagrams use the following notations to indicate required elements, cardinality, and containment:

- Required XML element
- ? Zero or one
- * Zero or more
- → Containment

Bean Namespace Diagram



Bean Namespace Elements

Element	Description
<alias>	Creates an alias for a bean definition.
<bean>	Defines a bean in the Spring container.
<constructor-arg>	Injects a value or a bean reference into an argument of the bean's constructor. Commonly known as constructor injection.
<description>	Used to describe a Spring context or an individual bean. Although ignored by the container, <description> can be used by tools that document Spring contexts.
<import>	Imports another Spring context definition.
<lookup-method>	Enables getter-injection by way of method replacement. Specifies a method that will be overridden to return a specific bean. Commonly known as getter-injection.
<meta>	Allows for meta-configuration of the bean. Only useful when there are beans configured that interprets and acts on the meta information.
<property>	Injects a value or a bean reference into a specific property of the bean. Commonly known as setter-injection.
<replaced-method>	Replaces a method of the bean with a new implementation.

The <bean> Element Distilled

Even though there are several XML elements that can be used to configure a Spring context, the one you'll probably use the most often is the <bean> element. Therefore, it only seems right that you get to know the attributes of <bean> in detail.

Attribute	Description
abstract	If **true**, the bean is abstract and will not be instantiated by the Spring container.
autowire	Declares how and if a bean should be autowired. Valid values are **byType**, **byName**, **constructor**, **autodetect**, or **no** for no autowiring.
autowire-candidate	If **false**, the bean is not a candidate for autowiring into another bean.
class	The fully-qualified class name of the bean.
dependency-check	Determines how Spring should enforce property setting on the bean. **simple** indicates that all primitive type properties should be set; **objects** indicates that all complex type properties should be set. Other value values are **default**, **none**, or **all**.
depends-on	Identifies a bean that should be instantiated by the container before this bean is instantiated.
destroy-method	Specifies a method that should be invoked when a bean is unloaded from the container.
factory-bean	Used with **factory-method**, specifies a bean that provides a factory method to create this bean.
factory-method	The name of a method that will be used instead of the constructor to instantiate this bean.
id	The identity of this bean in the Spring container.
init-method	The name of a method that should be invoked once the bean has been instantiated and injected.
lazy-init	If **true** the bean will be lazily instantiated. If **false**, the bean will be eagerly instantiated.
name	The name of the bean. This is a weaker alternative to **id**.
parent	Specifies a bean from whom this bean will inherit its configuration.
scope	Sets the scope of the bean. By default, all beans are **singleton**-scoped. Other scopes include **prototype**, **request**, and **session**.

Bean Namespace Example

The following Spring XML configures two beans, one injected into the other:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/
schema/beans
       http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd">

  <bean id="pirate" class="Pirate">
    <constructor-arg value="Long John Silver" />
    <property name="map" ref="treasureMap" />
  </bean>

  <bean id="treasureMap" class="TreasureMap" />
</beans>
```

The first bean is given "pirate" as its ID and is of type "Pirate." It is to be constructed through a constructor that takes a String as an argument—in this case, it will be constructed with "Long John Silver" as that value. In addition, its "map" property is wired with a reference to the "treasureMap" bean, which is defined as being an instance of TreasureMap.

Hot Tip

Don't put all your beans in one XML file. Once your application gets beyond the trivial stage, you'll likely have an impressive amount of XML in your Spring configuration. There's no reason to put all of that configuration in a single XML file. Keep your Spring configuration more manageable by splitting it across several XML files. Then assemble them all together when creating the application context or by using the <import> element:

```
<import resource="service-layer-config.xml" />
<import resource="data-layer-config.xml" />
<import resource="transaction-config.xml" />
```

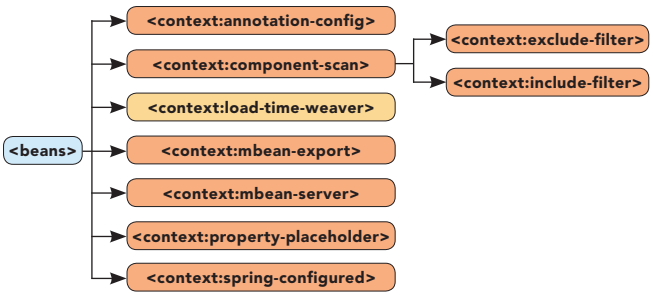
THE CONTEXT NAMESPACE

Schema URI
www.springframework.org/schema/context

Schema XSD
www.springframework.org/schema/context/spring-context-2.5.xsd

The context namespace was added in Spring 2.5 to provide several application context-specific configurations. It includes support for annotation-based configuration, JMX, and domain object injection.

Context Namespace Diagram



Context Namespace Elements

Element	Description
<context:annotation-config>	Enables annotation-based configuration in Spring beans. This element is not needed if the <context:component-scan> element is in use.
<context:component-scan>	Scans packages for beans to automatically register in the Spring container. Use of this element implies the same functionality as <context:annotation-config>.
<context:exclude-filter>	Used to exclude certain classes from being automatically registered by component-scan.
<context:include-filter>	Used to specify which classes to include when component-scan automatically registers beans.
<context:load-time-weaver>	Registers an AspectJ load-time weaver.
<context:mbean-export>	Exports beans as JMX MBeans.
<context:mbean-server>	Starts an MBean server with the Spring context.
<context:property-placeholder>	Enables external configuration via a properties file.
<context:spring-configured>	Enables injection into objects that are not instantiated by Spring.

Context Namespace Example

The following Spring configuration uses <context:component-scan> to automatically register certain beans from the "com.springinaction.service" namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/
beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/
context"
   xsi:schemaLocation="http://www.springframework.org/
schema/beans
       http://www.springframework.org/schema/beans/
spring-beans-2.5.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/
spring-context-2.5.xsd">

   <context:component-scan base-package="com.springinac-
tion.service" />

</beans>
```

As configured above, <context:component-scan> will scan the "com.springinaction.service" package and will automatically register as beans all of the classes it finds that are annotated with @Component, @Controller, @Repository, @Service, or @Aspect.

Hot Tip

Externalize configuration for end users

Not all configuration has to be done in Spring. You wouldn't expect the administrators or end users of your application to dig around in Spring XML to tweak database or other deployment-specific details, would you? Instead, externalize configuration using <context:property-placeholder>:

```
<context:property-placeholder
location="file:///etc/pirate.properties"
```

The name-value pairs from /etc/pirate.properties can then be used to fill in placeholder values in the Spring context. For example:

```
<bean id="pirate" class="Pirate">
  <constructor-arg value="${pirate.name}" />
</bean>
```

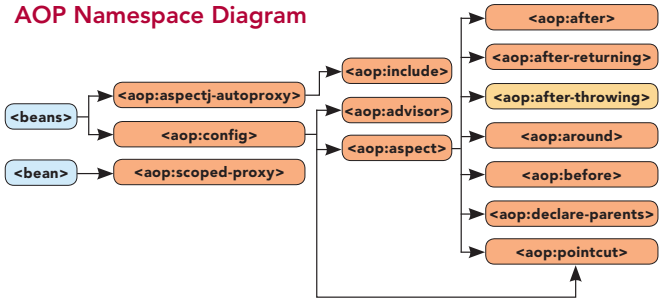
THE AOP NAMESPACE

Schema URI
www.springframework.org/schema/aop

Schema XSD
www.springframework.org/schema/aop/spring-aop-2.5.xsd

The aop namespace makes it possible to declare aspects, pointcuts, and advice in a Spring context. It also provides support for annotation-based aspects using @AspectJ annotations. Using aspects, you can define functionality that is applied (or "woven") across many points of your application.

AOP Namespace Diagram



AOP Namespace Elements

Element	Description
<aop:advisor>	Declares a Spring AOP advisor.
<aop:after>	Declares after advice (e.g., a method to be invoked after a pointcut).
<aop:after-returning>	Declares after-returning advice (e.g., a method to be invoked after a pointcut successfully returns).
<aop:after-throwing>	Declares after-throwing advice (e.g., a method to be invoked after an exception is thrown from a pointcut).
<aop:around>	Declares around advice (e.g., a method whose functionality wraps a pointcut).
<aop:aspect>	Defines an aspect, including one or more pointcuts and one or more advices.
<aop:aspectj-autoproxy>	Enables declaration of aspects using @AspectJ annotations.
<aop:before>	Declares before advice (e.g., a method to be invoked before a pointcut executes).
<aop:config>	The parent element for most elements in the AOP namespace.

AOP Namespace Elements, continued

Element	Description
<aop:declare-parents>	Defines an AOP introduction (effectively a mixin).
<aop:include>	Optionally used with aspectj-autoproxy to specify which @AspectJ-annotated beans to create proxies for.
<aop:pointcut>	Declares a pointcut (e.g., an opportunity for advice to be applied).
<aop:scoped-proxy>	Specifies a proxy for beans declared with complex scoping such as "request" and "session".

AOP Namespace Example

The following Spring configuration creates an aspect using elements from the aop namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:aop="http://www.springframework.org/schema/aop"
   xsi:schemaLocation="http://www.springframework.org/
schema/beans
       http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-
aop-2.5.xsd">

 <bean id="pirateTalker" class="PirateTalker" />

  <aop:config>
    <aop:pointcut id="plunderPointcut"
        expression="execution(* *.plunder(..))" />

    <aop:aspect ref="pirateTalker">
      <aop:before pointcut-ref="plunderPointcut"
          method="sayAvastMeHearties" />

      <aop:after-returning pointcut-ref="plunderPointcut"
          method="sayYarr" />
    </aop:aspect>
  </aop:config>
</beans>
```

The aspect is made up of one pointcut and two advice definitions. The pointcut is defined as the execution of the plunder() method on any object. The <aop:before> advice is configured to call the sayAvastMeHearties() method on the "pirateTalker" bean when the plunder() method is executed. Likewise, the sayYarr() method will be invoked upon execution of the plunder() method on any object.

Hot Tip

Reduce AOP-related XML by using @AspectJ annotations

The elements in the "aop" namespace make it rather easy to turn plain old Java objects into aspects. But the <aop:aspectj-autoproxy> element can single-handedly eliminate the need for almost all other "aop" namespace XML. By placing <aop:aspectj-autoproxy> in your Spring configuration, you can move your pointcut and advice declaration into your Java code using @AspectJ annotations such as @Aspect, @Pointcut, @Before, and @After. Refer to Chapter 4, section 4.3.2 of *Spring in Action, Second Edition* for more details.

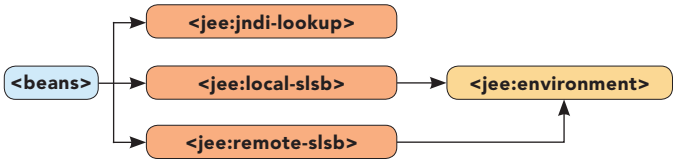
THE JEE NAMESPACE

Schema URI
www.springframework.org/schema/jee

Schema XSD
www.springframework.org/schema/jee/spring-jee-2.5.xsd

The JEE namespace provides configuration elements for looking up objects from JNDI as well as wiring references to EJBs into a Spring context.

JEE Namespace Diagram



JEE Namespace Elements

Element	Description
<jee:jndi-environment>	Defines environment settings for JNDI lookups.
<jee:jndi-lookup>	Declares a reference to an object to be retrieved from JNDI.
<jee:local-slsb>	Declares a reference to a local stateless session EJB.
<jee:remote-slsb>	Declares a reference to a remote stateless session EJB.

JEE Namespace Example

The following Spring configuration uses a few of the jee namespace's elements to retrieve objects from outside of Spring and configure them as Spring beans:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:jee="http://www.springframework.org/schema/jee"
   xsi:schemaLocation="http://www.springframework.org/
schema/beans
       http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd
       http://www.springframework.org/schema/jee
       http://www.springframework.org/schema/jee/spring-
jee-2.5.xsd">

   <jee:remote-slsb id="hispaniola"
      jndi-name="ejb/PirateShip"
      business-interface="com.pirate.PirateShipEjb"
      resource-ref="true" />

   <jee:jndi-lookup id="parrot"
      jndi-name="pirate/Parrot "
      resource-ref="false" />

</beans>
```

The first element, <jee:remote-slsb>, configures a bean named "Hispaniola" which is actually a reference to an EJB 2 remote stateless session bean. The EJB's home interface is found in JNDI under the name "java:comp/env/ejb/PirateShip". The resource-ref attribute indicates that the value in jndi-name should be prefixed by "java:comp/env/". The EJB implements methods defined in the PirateShipEjb business interface.

The other element, <jee:jndi-lookup>, retrieves a reference to an object from JNDI (it could be an EJB 3 session bean or just a plain Java object). The object is found in JNDI under the name "pirate/Parrot". Because resource-ref is "false", the jndi-name is not prefixed with "java:comp/env/".

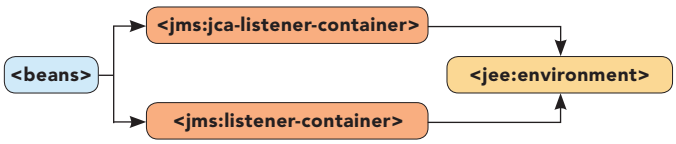
THE JMS NAMESPACE

Schema URI
www.springframework.org/schema/jms

Schema XSD
www.springframework.org/schema/jms/spring-jms-2.5.xsd

The JMS namespace provides elements for configuring message-driven POJOs, beans that respond to messages that arrive on a JMS destination (either a topic or a queue).

JMS Namespace Diagram



JMS Namespace Elements

Element	Description
<jms:jca-listener-container>	Configures a container for JCA-based JMS destination listeners.
<jms:listener-container>	Configures a container for standard JMS destination listeners.
<jms:listener>	Declares a listener to a JMS destination. Used to create message-driven POJOs.

JMS Namespace Example

The following Spring configuration sets up a message-driven POJO that responds to messages that arrive on a queue.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:jms="http://www.springframework.org/schema/jms"
   xsi:schemaLocation="http://www.springframework.org/schema/
beans
       http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd
       http://www.springframework.org/schema/jms
       http://www.springframework.org/schema/jms/spring-jms-
2.5.xsd">
…
  <bean id="connectionFactory"
      class="org.apache.activemq.ActiveMQConnectionFactory">
    <property name="brokerURL" value="tcp://localhost:61616"
/>
  </bean>

  <bean id="messageHandlerService" class="com.pirate.Message-
HandlerImpl" />

  <jms:listener-container connection-
factory="connectionFactory">
    <jms:listener
        destination="queue.bottle"
        ref="messageHandlerService"
        method="readMessageFromBottle" />
  </jms:listener-container>
</beans>
```

The <jms:listener-container> configures a container for handling messages arriving on topics or queues coming in on the JMS connection factory. Within this element you may declare one or more <jms:listener> elements to respond to specific topics. In this case, the single <jms:listener> reacts to messages arriving in the "queue.bottle" topic, invoking the readMessage-FromBottle() method of the "messageHandlerService" bean when they arrive.

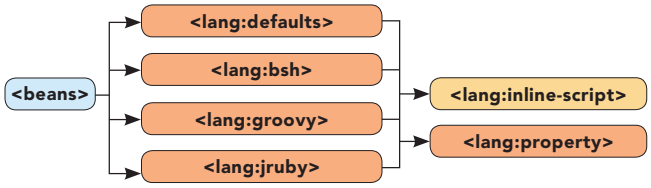
THE LANG NAMESPACE

Schema URI
www.springframework.org/schema/lang

Schema XSD
www.springframework.org/schema/lang/spring-lang-2.5.xsd

The "lang" namespace enables you to wire scripted objects into Spring. These objects can be written in either Groovy, JRuby, or BeanShell.

Lang Namespace Diagram



Lang Namespace Elements

Element	Description
<lang:bsh>	Configures a BeanShell-defined bean.
<lang:defaults>	Configures defaults to be applied to all scripted beans.
<lang:groovy>	Declares a bean implemented as a Groovy script.
<lang:inline-script>	Embeds a scripted bean's code directly in Spring XML.
<lang:jruby>	Declares a bean implemented as a JRuby script.
<lang:property>	Used to inject values or references into scripted beans.

Lang Namespace Example

In this Spring context, a Pirate bean is injected with scripted beans defined with <lang:groovy> and <lang:jruby>:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:lang="http://www.springframework.org/schema/lang"
   xsi:schemaLocation="http://www.springframework.org/
schema/beans
       http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd
       http://www.springframework.org/schema/lang
       http://www.springframework.org/schema/lang/spring-
lang-2.5.xsd">

   <bean id="jackSparrow" class="Pirate">
    <constructor-arg value="Jack Sparrow" />
    <property name="compass" ref="compass" />
    <property name="hat" ref="hat" />
   </bean>

   <lang:groovy id="compass"
      script-source="classpath:Compass.groovy"
      refresh-check-delay="10000" />

   <lang:jruby id="hat"
      script-source="classpath:PirateHat.rb"
      script-interface="PirateHat"
      refresh-check-delay="60000" />
</beans>
```

The <lang:groovy> element creates a bean that is implemented as a Groovy script called Compass.groovy and found in the root of the classpath. The refresh-check-delay attribute indicates that the script should be checked every 10 seconds for updates and reloaded if the script changes.

The <lang:jruby> element creates a bean that is implemented as a Ruby (JRuby, specifically) script called PirateHat.rb. It implements a PirateHat interface and is checked for updates once per minute.

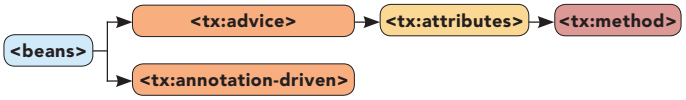
THE TX NAMESPACE

Schema URI
www.springframework.org/schema/tx

Schema XSD
www.springframework.org/schema/tx/spring-tx-2.5.xsd

The "tx" namespace provides support for declarative transactions across beans declared in Spring.

TX Namespace Diagram



TX Namespace Elements

Element	Description
<tx:advice>	Declares transactional advice.
<tx:annotation-driven>	Configures Spring to use the @Transactional annotation for transactional rules.
<tx:attributes>	Declares transactional rules for one or more methods.
<tx:jta-transaction-manager>	Configures a JTA transaction manager, automatically detecting WebLogic, WebSphere, or OC4J.
<tx:method>	Describes transactional rules for a given method signature.

TX Namespace Example

The following Spring configuration uses elements in the tx namespace to configure transactional rules and boundaries:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:tx="http://www.springframework.org/schema/tx"
   xmlns:aop="http://www.springframework.org/schema/aop"
   xsi:schemaLocation="http://www.springframework.org/
schema/beans
       http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-
aop-2.5.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx-
2.5.xsd">

  <tx:jta-transaction-manager />

  <tx:advice id="txAdvice">
    <tx:attributes>
      <tx:method name="plunder*" propagation="REQUIRED" />
      <tx:method name="*" propagation="SUPPORTS" />
    </tx:attributes>
  </tx:advice>

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

The <tx:jta-transaction-manager> was added in Spring 2.5 to automatically detect the JTA transaction manager provided by either WebLogic, WebSphere, or OC4J. It exposes the transaction manager as a bean in the Spring context with the name "transactionManager".

Next, the <tx:advice> sets up AOP advice that declares the transactional rules. In this case, any methods with names that start with "plunder" require transactions. All other methods support transactions, but do not require them. Finally, this example borrows from the aop namespace to configure an AOP advisor that uses the transactional advice. The pointcut here is for all methods in the Pirate class.

Hot Tip

Configure transactional rules in Java

If you're looking for ways to cut back on the amount of XML in a Spring configuration, consider using the <tx:annotation-driven> element. Once this element is in place, you can start annotating your beans and their methods with @Transactional to define transactional boundaries and rules. Have a look at chapter 6, section 6.4.4 of *Spring in Action, Second Edition* to learn more.

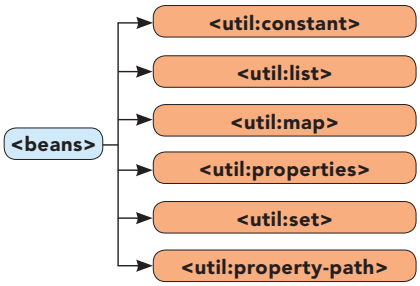
THE UTIL NAMESPACE

Schema URI
www.springframework.org/schema/util

Schema XSD
www.springframework.org/schema/util/spring-util-2.5.xsd

The utility namespace provides elements that make it possible to wire collections and other non-bean objects in Spring as if they were any other bean.

Util Namespace Diagram



Util Namespace Elements

Element	Description
<util:constant>	References a static field on a type and exposes its value as a bean.
<util:list>	Declares a list of values or references as a bean.
<util:map>	Declares a map as a bean.
<util:properties>	Loads a java.util.Properties from a properties file and exposes it as a bean.
<util:set>	Declares a set as a bean.
<util:property-path>	References a bean property (or a nested property) and exposes that property as a bean itself.

Util Namespace Example

The following Spring configuration uses several elements from the "util" namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:util="http://www.springframework.org/schema/util"
   xsi:schemaLocation="http://www.springframework.org/
schema/beans
       http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd
       http://www.springframework.org/schema/util
       http://www.springframework.org/schema/util/spring-
util-2.5.xsd">
```

THE UTIL NAMESPACE, *continued*

Util Namespace Example, continued

```
  <util:list id="piratePhrases">
    <value>Yo ho ho</value>
    <value>Yarr</value>
    <value>Avast me hearties!</value>
    <value>Blow me down</value>
  </util:list>

  <util:constant id="pirateCode"
      static-field="Pirate.PIRATE_CODE" />

  <util:property-path id="doubloonCount"
      path="pirate.treasure.doubloonCount" />

</beans>
```

The <util:list> element is used here to create a list of Strings containing various phrases uttered by pirates. The <util:constant> element creates a reference to the constant (public static field) named PIRATE_CODE of the Pirate class. Finally, the <util:property-path> digs deep into the "pirate" bean, retrieving the value of the doubloonCount property of the treasure property of the bean named "pirate". In all three cases, the resulting values are exposed as beans in the Spring context, suitable for injection into other beans.

SPRING ANNOTATIONS

Historically, Spring configuration has primarily involved XML. But that is changing as Spring gradually embraces annotation-driven configuration. As of Spring 2.5, there are 36 annotations provided by Spring, not to mention annotations provided by third party libraries and various Spring add-ons.

Context Configuration Annotations

These annotations are used by Spring to guide creation and injection of beans.

Annotation	Use	Description
@Autowired	Constructor, Field, Method	Declares a constructor, field, setter method, or configuration method to be autowired by type. Items annotated with @Autowired do not have to be public.
@Configurable	Type	Used with <context:spring-configured> to declare types whose properties should be injected, even if they are not instantiated by Spring. Typically used to inject the properties of domain objects.
@Order	Type, Method, Field	Defines ordering, as an alternative to implementing the org.springframework.core. Ordered interface.
@Qualifier	Field, Parameter, Type, Annotation Type	Guides autowiring to be performed by means other than by type.
@Required	Method (setters)	Specifies that a particular property must be injected or else the configuration will fail.
@Scope	Type	Specifies the scope of a bean, either singleton, prototype, request, session, or some custom scope.

Transaction Annotations

The @Transactional annotation is used along with the <tx:annotation-driven> element to declare transactional boundaries and rules as class and method metadata in Java.

Annotation	Use	Description
@Transactional	Method, Type	Declares transactional boundaries and rules on a bean and/or its methods.

Stereotyping Annotations

These annotations are used to stereotype classes with regard to the application tier that they belong to. Classes that are annotated with one of these annotations will automatically be registered in the Spring application context if <context: component-scan> is in the Spring XML configuration.

In addition, if a PersistenceExceptionTranslationPostProcessor is configured in Spring, any bean annotated with @Repository will have SQLExceptions thrown from its methods translated into one of Spring's unchecked DataAccessExceptions.

Annotation	Use	Description
@Component	Type	Generic stereotype annotation for any Spring-managed component.
@Controller	Type	Stereotypes a component as a Spring MVC controller.
@Repository	Type	Stereotypes a component as a repository. Also indicates that SQLExceptions thrown from the component's methods should be translated into Spring DataAccessExceptions.
@Service	Type	Stereotypes a component as a service.

Spring MVC Annotations

These annotations were introduced in Spring 2.5 to make it easier to create Spring MVC applications with minimal XML configuration and without extending one of the many implementations of the Controller interface.

Annotation	Use	Description
@Controller	Type	Stereotypes a component as a Spring MVC controller.
@InitBinder	Method	Annotates a method that customizes data binding.
@ModelAttribute	Parameter, Method	When applied to a method, used to preload the model with the value returned from the method. When applied to a parameter, binds a model attribute to the parameter.
@RequestMapping	Method, Type	Maps a URL pattern and/or HTTP method to a method or controller type.
@RequestParam	Parameter	Binds a request parameter to a method parameter.
@SessionAttributes	Type	Specifies that a model attribute should be stored in the session.

JMX Annotations

These annotations, used with the <context:mbean-export> element, declare bean methods and properties as MBean operations and attributes.

Annotation	Use	Description
@ManagedAttribute	Method	Used on a setter or getter method to indicate that the bean's property should be exposed as an MBean attribute.
@ManagedNotification	Type	Indicates a JMX notification emitted by a bean.
@ManagedNotifications	Type	Indicates the JMX notifications emitted by a bean.
@ManagedOperation		Specifies that a method should be exposed as an MBean operation.
@ManagedOperation Parameter		Used to provide a description for an operation parameter.
@ManagedOperation Parameters		Provides descriptions for one or more operation parameters.
@ManagedResource	Type	Specifies that all instances of a class should be exposed as MBeans.

SPRING ANNOTATIONS, *continued*

Testing Annotations

These annotations are useful for creating unit tests in the JUnit 4 style that depend on Spring beans and/or require a transactional context.

Annotation	Use	Description
@AfterTransaction	Method	Used to identify a method to be invoked after a transaction has completed.
@BeforeTransaction	Method	Used to identify a method to be invoked before a transaction starts.
@ContextConfiguration	Type	Configures a Spring application context for a test.
@DirtiesContext	Method	Indicates that a method dirties the Spring container and thus it must be rebuilt after the test completes.
@ExpectedException	Method	Indicates that the test method is expected to throw a specific exception. The test will fail if the exception is not thrown.
@IfProfileValue	Type, Method	Indicates that the test class or method is enabled for a specific profile configuration.
@NotTransactional	Method	Indicates that a test method must not execute in a transactional context.
@ProfileValueSource Configuration	Type	Identifies an implementation of a profile value source. The absence of this annotation will cause profile values to be loaded from system properties.
@Repeat	Method	Indicates that the test method must be repeated a specific number of times.
@Rollback	Method	Specifies whether or not the transaction for the annotated method should be rolled back or not.
@TestExecution Listeners	Type	Identifies zero or more test execution listeners for a test class.
@Timed	Method	Specifies a time limit for the test method. If the test does not complete before the time has expired, the test will fail.
@Transaction Configuration	Type	Configures test classes for transactions, specifying the transaction manager and/or the default rollback rule for all test methods in a test class.

ABOUT THE AUTHOR

Craig Walls

Craig Walls is a Texas-based software developer with more than 13 years' experience working in the telecommunication, financial, retail, educational, and software industries. He's a zealous promoter of the Spring Framework, speaking frequently at local user groups and conferences and writing about Spring on his blog. When he's not slinging code, Craig spends as much time as he can with his wife, two daughters, six birds, three dogs, and an ever-fluctuating number of tropical fish.

Publications

- *Spring in Action, 2nd Edition, 2007*
- *XDoclet in Action, 2003*

Blog

- http://www.springinaction.com

Projects

- Committer to XDoclet project; Originator of Portlet and Spring modules for XDoclet

RECOMMENDED BOOK

Spring in Action, 2nd Edition is a practical and comprehensive guide to the Spring Framework, the framework that forever changed enterprise Java development. What's more, it's also the first book to cover the new features and capabilities in Spring 2.

BUY NOW

books.dzone.com/books/spring-in-action

DZone

The DZone Network is a group of free online services that aim to satisfy the information needs of software developers and architects. From news, blogs, tutorials, source code and more, DZone offers everything technology professionals need to succeed.

To quote *PC magazine*, "DZone is a developer's dream."

DZone, Inc.
1251 NW Maynard
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-05-9
ISBN-10: 1-934238-05-8
50795
9 781934 238059
$7.95